Programming Assignment #2

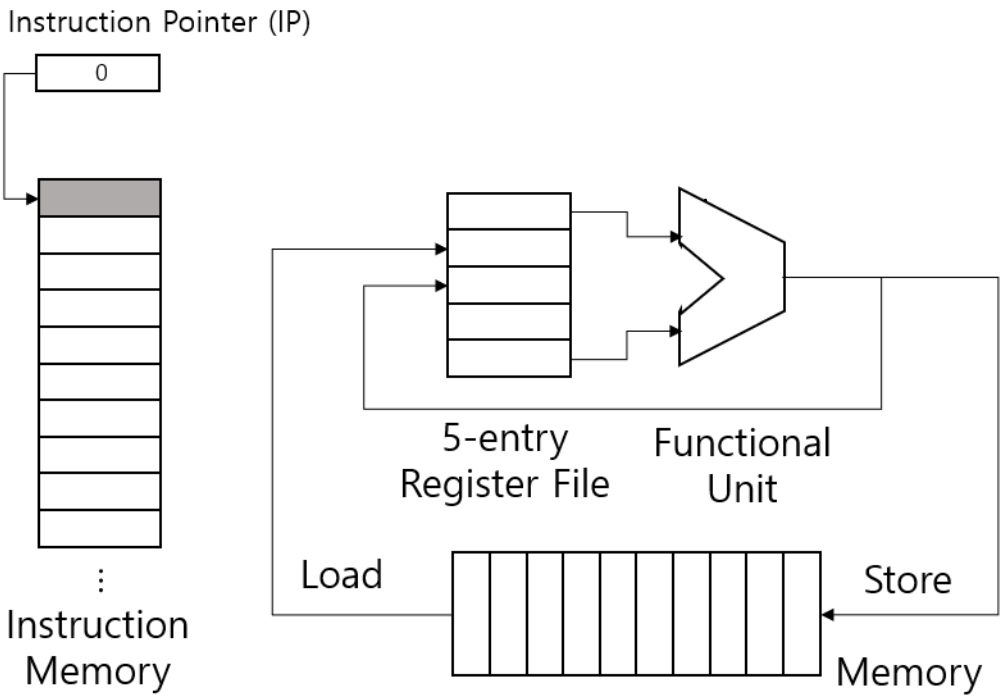
Simple Emulator

이번 과제에서는 파이썬을 활용하여 **SimpleCPU**를 에뮬레이션하는 에뮬레이터, emulator.py를 만드는 것입니다.

emulator.py는 다음과 같이 SimpleCPU의 명령어들이 적힌 텍스트 파일을 받습니다.

```
Usage: ./emulator.py program.txt
```

SimpleCPU의 사양은 다음과 같습니다.



- 데이터는 Float/Int 타입이 저장된다.
- 데이터를 10개 저장할 수 있는 메모리
- 데이터를 5개 저장할 수 있는 레지스터파일
- 명령어를 저장하는 메모리는 데이터 메모리와 별도로 있으며 무한하다고 가정
- 프로그램 시작 시, 모든 레지스터 값은 0으로 시작한다.
- 계산 방식은 다음과 같습니다.
  - 초기 데이터는 전부 메모리에 있음

- 명령어를 사용하여 계산하기 위해서는 레지스터 파일에 데이터를 ld 명령어를 사용하여 올려야 한다.
- 데이터를 메모리에 저장하기 위해서는 st 명령어를 사용한다.
- IP에 저장된 인덱스 위치의 명령어를 실행한다. 명령어 실행 후, IP는 1증가하여, 다음 명령어를 실행할 수 있도록 한다.
- 브랜치 명령어를 만날 경우, 조건에 따라 IP값을 수정한다.
- 계산 명령어는 레지스터 파일의 데이터만 접근이 가능하다.
- 계산 순서는 주어진 파일에 기록된 명령어 순서대로 처리한다. 브랜치 명령어를 만날 경우,

레지스터 파일은 $0, $2, ..., $4로 접근이 가능하다. 메모리는 0, 2, 3, ..., 9으로 명시를 한다.

SimpleCPU가 지원하는 명령어는 아래의 표에 있습니다. 명령어 설명에서 <left> <- <right> 의 의미는 <right>의 결과를 <left>에 저장한다는 의미입니다.

- 계산 명령

명령어 포멧	설명	예제
add <dst>, <src1>, <src2>	dst <- src1 + src2	add $1, $2, $3
sub <dst>, <src1>, <src2>	dst <- src1 - src2	sub $1, $2, $3
div <dst>, <src1>, <src2>	dst <- src1 / src2	div $1, $2, $3
mul <dst>, <src1>, <src2>	dst <- src1 * src2	mul $1, $2, $3

- 메모리 명령어

명령어 포멧	설명	예제
ld <dst reg>, <memory>	<dst reg> <- <memory>	ld $1, 10
st <src reg>, <memory>	<memory> <- <src reg>	st $2, 5

- 브랜치 명령어

명령어 포멧	설명	예제
jump <inst idx>	IP <- <inst idx>	jmp 2
beq <src1>, <src2>, <inst idx>	IP <- <inst idx> if <src1> == <src2>	beq $1, $2, 2
ble <src1>, <src2>, <inst idx>	IP <- <inst idx> if <src1> <= <src2>	ble $1, $2, 2
Bne <src1>, <src2>, <inst idx>	IP <- <inst idx> If <src1> != <src2>	bneq $1, $2, 2

emulator.py는 입력받은 명령어를 다 계산한 후, CPU의 최종 상태를 출력해야 한다.

출력은 다음과 같아야 한다. (...은 문서상 생략한 부분으로 실제로는 전부 출력을 하여야 함., <>은 메모리나 레지스터에 기록된 숫자 값임 출력 시 '<', '>'는 포함하지 않음)

```
IP: <실행이 완료된 후, IP의 값>
Register File:
$0: <reg value1>
$1: <reg value2>
$2: <reg value3>
$3: <reg value4>
$4: <reg value5>
Memory:
[0]: <value1>
[1]: <value2>
...
[9]: <value10>
```

입력 받는 프로그램의 구성은 다음과 같다.

리스트 표현 형식으로 10개의 메모리 값을 받는다. 이 후, 명령어들을 순서대로 기록한다. 프로그램의 명령어 수는 제한이 없다.

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,]
명령어 1
명령어 2
...
```

테스트를 위한 예제 프로그램 (example.txt)

Memory[0], Memory[1]이 가리키는 범위의 값을 순차적으로 더하고 최종 결과는 Memory[2]에 기록하는 프로그램

```
[1, 10, 0, 0, 0, 0, 0, 0, 0, 0]
ld $0, 0
ld $1, 0
ld $2, 1
add $3, $3, $1
add $1, $1, $0
ble $1, $2, 3
st $3, 2
```

예상 출력 결과

```
IP: 7
Register File:
$0: 1
$1: 11
$2: 10
$3: 55
$4: 0
Memory:
[0]: 1
[1]: 10
[2]: 55
[3]: 0
[4]: 0
[5]: 0
[6]: 0
[7]: 0
[8]: 0
[9]: 0
```

구현 제약 사항은 다음과 같습니다.

다음의 Exception을 정의하세요.

SimpleCPU_REGIndexError, SimpleCPU_IPIndexError, SimpleCPU_MEMIndexError

각각 레지스터와 명령어 IP, 메모리 접근 범위를 명령어가 위반할 경우, 해당되는 Exception을 발생시키도록 합니다.

Exception이 발생할 경우, 다음의 메시지를 출력하고 프로그램을 종료하도록 합니다.

<Exception 이름> | IP:<Exception을 발생시킨 명령어의 IP | <명령어> | <에러를 발생시킨 인덱스> | 0, <접근하는 데이터의 익덱스의 범위>

첫번째 명령어 add $6, $7, $9가 register의 범위를 넘을 경우, 이 때 범위를 벗어난 인덱스가 다수일 경우, 명령어 인자 순서가 가장 빠른 것을 출력한다. 여기서는 $6이 된다.

SimpleCPU_REGIndexError | IP:1 | add $6, $7, $9 | 6 | 0, 4

IP나 메모리 인덱스가 위반될 경우에는, 위반되는 인덱스 값과 접근하는 데이터 (명령어 또는 메모리)의 최대 접근 가능 범위를 출력한다.

LD/ST/Branch 명령어의 경우, 레지스터와 메모리, 또는 레지스터와 명령어 메모리를 접근하기 때문에 동시에 두 가지 에러가 발생할 수 있다. 이 경우, 먼저 발견되는 에러에 관해 exception을 발생시키고 에러메시지와 함께 종료하면 됩니다. 과제에서는 명령어 분석 시, 인자를 왼쪽에서부터 읽어 처리를 하므로, 인자 기준 왼편에서 발생한 에러를 다루면 됩니다.